Pointers

```
#include<iostream>
using std::cout;
using std::endl;

int f(int *r){
        *r += 3;
        return *r;
}

int main(){
        int x;
        int *p;
        x = 5;
        p = &x;

        cout << "p is: "  << p << endl;          // 0x6ffe04
        cout << "&x is: " << &x << endl;         // 0x6ffe04
        cout << "*p is: " << *p << endl;         // 5
        cout << "x is: "  << x << endl;          // 5

        cout << "f(p) is: " << f(p) << endl;     // 8
        cout << "p is: "    << p << endl;                // 0x6ffe04
        cout << "&x is: "   << &x << endl;               // 0x6ffe04
        cout << "*p is: "   << *p << endl;               // 8
        cout << "x is: "    << x << endl;                // 8

        cout << "f(&x) is: " << f(&x) << endl;   // 11
        cout << "p is: "     << p << endl;                // 0x6ffe04
        cout << "&x is: "    << &x << endl;               // 0x6ffe04
        cout << "*p is: "    << *p << endl;               // 11
        cout << "x is: "     << x << endl;                // 11

        return 0;
}


/*
int main(){
        int x;
        int *p = &x;
        x = 5;
        cout << "p is: "  << p << endl;          // 0x6ffe04
        cout << "&x is: " << &x << endl;         // 0x6ffe04
        cout << "*p is: " << *p << endl;         // 5
        cout << "x is: "  << x << endl;          // 5
        return 0;
}
*/


/*
int main(){
        int x;
        int *p;
        x = 5;
        p = &x;
        cout << "p is: "  << p << endl;          // 0x6ffe04
        cout << "&x is: " << &x << endl;         // 0x6ffe04
        cout << "*p is: " << *p << endl;         // 5
        cout << "x is: "  << x << endl;          // 5
        return 0;
}
*/
```

this means: we're expecting a pointer (an address)

"this"

```cpp
//-----------------------------------------------------------------------------------------------------------------------
// main.cpp
//-----------------------------------------------------------------------------------------------------------------------
#include <iostream>
#include "X.h"

int main() {
    X obj(3);
    std::cout << "Hello, World!" << std::endl;
    std::cout << obj.getMember() << std::endl;
    return 0;
}
```

```cpp
//-----------------------------------------------------------------------------------------------------------------------
// X.h
//-----------------------------------------------------------------------------------------------------------------------
#ifndef X_H
#define X_H

class X {
    int member;

public:
    X(int member);

public:
    int getMember() const;
    void setMember(int member);
};

#endif //X_H
```

```cpp
//-----------------------------------------------------------------------------------------------------------------------
// X.cpp
//-----------------------------------------------------------------------------------------------------------------------
#include "X.h"

// Getter
int X::getMember() const {
    return member;
}

// Setter
void X::setMember(int member) {
    this->member = member;              // Here we cannot have: " member = member;" because of the scope.
}                                       // The local variable is stronger so this wouldn't be an assignment.
                                        // "this" stores the address of the object we run "setMember(..)" on.
                                        // Without "this" the proper notation is: (an alternative)
                                        //              void X::setMember(int member){
                                        //                      X::member = member;

// Constructor
X::X(int member){                       // an alternative to the version with "this" is:
    this->member=member;                //              X::X(int member): member(member) {}
}
```

MatlabVector (pt. 1)

```
//-----------------------------------------------------------------------------------------------------------------
// main.cpp
//-----------------------------------------------------------------------------------------------------------------
#include <iostream>
#include "MatlabVector.h"

using std::cout;
using std::endl;

int main() {
    MatlabVector v;
    v.set(0, 1);
    v.set(1, 3);
    std::cout << "v content" << std::endl;
    v.print();
    v.set(3,4);
    std::cout << "v content" << std::endl;
    v.print();

    double d = v.get(4);
    std::cout << "v content" << std::endl;
    v.print();

    for (unsigned i = 0; i < v.size(); ++i) { // pretty ugly:
        v.set(i,i);
        cout << v.get(i) << " ";
    }
    cout << endl;

    std::cout << "v content" << std::endl;
    v.print();

    MatlabVector v2 = v *3;// unfortunately 3*v does not work
    std::cout << "v2 content" << std::endl;
    v2.print();
    MatlabVector v3 = v+v2;
    std::cout << "v3 content" << std::endl;
    v3.print();

    return 0;
}
```

We don't like it like this.
We would like to have:

```
for(unsigned i=0; i < v.size(); ++i) {
    v[i] = i;
    cout << v[i];
}
```

```
//-----------------------------------------------------------------------------------------------------------------
// MatlabVector.h
//-----------------------------------------------------------------------------------------------------------------
#ifndef MATLABVECTOR_MATLABLIKEVECTOR_H
#define MATLABVECTOR_MATLABLIKEVECTOR_H

#include <vector>
#include <iostream>

class MatlabVector {

    std::vector<double> elem;

public:

    double get(unsigned n);                 // access: read
    void set(unsigned n, double v);         // access: write
    size_t size() const;                    // return number of elements

    void print() const;

    MatlabVector operator+(const MatlabVector& other) const;
    MatlabVector operator*(double scalar) const;
};

#endif //MATLABVECTOR_MATLABLIKEVECTOR_H
```

```cpp
//-----------------------------------------------------------------------------
// MatlabVector.cpp
//-----------------------------------------------------------------------------
#include "MatlabVector.h"

using std::cout;
using std::endl;

void MatlabVector::set(unsigned n, double v){
    while (elem.size() < n+1)
        elem.push_back(0.);
    elem[n] = v;
}

double MatlabVector::get(unsigned n){
    while (elem.size() < n+1)
        elem.push_back(0.);
    return elem[n];
}

void MatlabVector::print() const{
    for (size_t i =0; i< elem.size(); ++i)
        cout << elem[i] << " ";
    cout << endl;
}

size_t MatlabVector::size() const{
    return elem.size();
}

MatlabVector MatlabVector::operator*(double scalar) const{
    MatlabVector result;
    for (unsigned i=0; i<elem.size(); ++i)
        result.set(i, scalar * elem[i]);
    return result;
}

MatlabVector MatlabVector::operator+(const MatlabVector &other) const{
    MatlabVector result;
    for (unsigned i=0; i<elem.size(); ++i)
        result.set(i,elem[i] + other.elem[i]);
    return result;
}
```

MatlabVector (pt. 2)

```
//---------------------------------------------------------------------------------------------------------------------
// main.cpp
//---------------------------------------------------------------------------------------------------------------------
#include <iostream>
#include "MatlabVector.h"

using std::cout;
using std::endl;

int main() {
    MatlabVector v;
    v[0] = 1;
    v[1] = 3;
    cout << "v content" << endl;
    v.print();
    v[3] = 4;
    cout << "v content" << endl;
    v.print();

    double d = v[4];
    cout << "v content" << endl;
    v.print();

    for (unsigned i=0; i<v.size(); ++i) {
        v[i] = i;
        cout << v[i] << " ";
    }
    cout << endl;

    cout << "v content" << endl;
    v.print();

    MatlabVector v2 = v * 3; // unfortunately 3*v does not work
    cout << "v2 content" << endl;
    v2.print();

    MatlabVector v3 = v + v2;
    cout << "v3 content" << endl;
    v3.print();

    return 0;
}


//---------------------------------------------------------------------------------------------------------------------
// MatlabVector.h
//---------------------------------------------------------------------------------------------------------------------
#ifndef MATLABVECTOR_MATLABLIKEVECTOR_H
#define MATLABVECTOR_MATLABLIKEVECTOR_H

#include <vector>
#include <iostream>

class MatlabVector {

    std::vector<double> elem;

public:

    double & operator[](unsigned n);
    size_t size() const; // return number of elements

    void print() const;

    MatlabVector operator+(const MatlabVector& other) const;
    MatlabVector operator*(double scalar) const;
};

#endif //MATLABVECTOR_MATLABLIKEVECTOR_H
```

```cpp
//-----------------------------------------------------------------------------
// MatlabVector.cpp
//-----------------------------------------------------------------------------
#include "MatlabVector.h"

using std::cout;
using std::endl;

void MatlabVector::print() const {
    for (size_t i =0; i< elem.size(); ++i)
        cout << elem[i] << " ";
    cout << endl;
}

size_t MatlabVector::size() const{
    return elem.size();
}

MatlabVector MatlabVector::operator*(double scalar) const{
    MatlabVector result;
    for (unsigned i=0; i<elem.size(); ++i)
        result[i] =scalar * elem[i];
    return result;
}

MatlabVector MatlabVector::operator+(const MatlabVector &other) const {
    MatlabVector result;
    for (unsigned i=0; i<elem.size(); ++i)
        result[i] = elem[i] + other.elem[i];
    return result;
}

double & MatlabVector::operator[](unsigned int n) {
    while (elem.size() < n+1)
        elem.push_back(0.);
    return elem[n];
}
```

Delegating Constructors

```
//-----------------------------------------------------------------------------------------------------------------------
// main.cpp
//-----------------------------------------------------------------------------------------------------------------------
#include <iostream>
#include "Sales_data.h"

int main() {

  std::cout << "This is s1"<<std::endl;
  Sales_data s1("01",1,5);
  s1.print();

  std::cout << "This is s2"<<std::endl;
  Sales_data s2("02");
  s2.print();

  std::cout << "This is s3"<<std::endl;
  Sales_data s3;
  s3.print();

}
```

Output:
This is s1
3 parameters version
ISBN: 01 unit solds: 1 revenue: 5

output:
This is s2
3 parameters version
1 parameter version
ISBN: 02 unit solds: 0 revenue: 0

output:
This is s3
3 parameters version
Default version
ISBN: unit solds: 0 revenue: 0

```
//-----------------------------------------------------------------------------------------------------------------------
// Sales_data.h
//-----------------------------------------------------------------------------------------------------------------------
#ifndef SALES_DATA_H
#define SALES_DATA_H
#include <iostream>

class Sales_data {
public:

        // constructors (the 2nd and 3rd delegate on the first)
        Sales_data(std::string s, unsigned cnt, double price): bookNo(s), units_sold(cnt), revenue(cnt*price)
                { std::cout << "3 parameters version\n"; }

        Sales_data(): Sales_data("", 0, 0) { std::cout << "Default version\n"; }

        Sales_data(std::string s): Sales_data(s, 0,0) { std::cout << "1 parameter version\n"; }

        std::string isbn() const { return bookNo; }
        void print () const;

private:
        std::string bookNo ;
        unsigned units_sold ;
        double revenue ;
};

#endif


//-----------------------------------------------------------------------------------------------------------------------
// Sales_data.cpp
//-----------------------------------------------------------------------------------------------------------------------
#include "Sales_data.h"

void Sales_data::print () const{
  std::cout << "ISBN: " << bookNo << " unit solds: " << units_sold << " revenue: " << revenue <<std::endl;
}
```

Sneaky

```
//------------------------------------------------------------------------------------------------------------------
// Base.h
//------------------------------------------------------------------------------------------------------------------
#ifndef SNEAKY_BASE_H
#define SNEAKY_BASE_H

class Base {
protected:
    int prot_mem = 0;
public:
    int getProtMem() const;
    void setProtMem(int protMem); // protected member
};

#endif //SNEAKY_BASE_H


//------------------------------------------------------------------------------------------------------------------
// Sneaky.h
//------------------------------------------------------------------------------------------------------------------
#ifndef SNEAKY_SNEAKY_H
#define SNEAKY_SNEAKY_H

#include "Base.h"

class Sneaky : public Base {
    void clobber1(Sneaky&); // can access Sneaky::prot_mem
    void clobber2(Base&);   // can't access Base::prot_mem
    int j;                  // j is private by default
};

#endif //SNEAKY_SNEAKY_H


//------------------------------------------------------------------------------------------------------------------
// Base.cpp
//------------------------------------------------------------------------------------------------------------------
#include "Base.h"

int Base::getProtMem() const {
    return prot_mem;
}

void Base::setProtMem(int protMem) {
    prot_mem = protMem;
}


//------------------------------------------------------------------------------------------------------------------
// Sneaky.cpp
//------------------------------------------------------------------------------------------------------------------
#include "Sneaky.h"

// ok: clobber1 can access the private and protected members in Sneaky objects
void Sneaky::clobber1(Sneaky &s) { s.j = s.prot_mem = 0; }

// error: clobber can't access the protected members in Base@void
Sneaky::clobber2(Base &b) { b.prot_mem = 0; }
```